Técnicas e Algoritmos de Compressão Não-Destrutiva de Imagens Monocromáticas

Eduardo F. Ferreira Cruz
Departamento de Engenharia Informática
Universidade de Coimbra
eduardo.cruz@student.uc.pt

Gonçalo Marinho Barroso
Departamento de Engenharia Informática
Universidade de Coimbra
uc2019216314@student.uc.pt

Inês Ferreira Mendes
Departamento de Engenharia Informática
Universidade de Coimbra
inesmendes@student.dei.uc.pt

***Resumo*—Da rápida evolução dos computadores e da Internet, e com o consequente aumento do uso de aplicações e conteúdo multimédia, entre outros, cujo tamanho em memória requer uma capacidade de armazenamento e de transmissão em rede elevada, e dado que a memória do computador é limitada, assim como a largura de banda em rede, surge a necessidade de comprimir os dados dos ficheiros, ou seja, de reduzir o espaço em memória ocupado pelos mesmos. Neste artigo apresentamos e analisamos vários algoritmos amplamente utilizados nas técnicas tradicionais e modernas de compressão não-destrutiva de dados que, tal como o nome sugere, não implicam a perda/destruição de dados, ou seja, a partir dos dados codificados é possível obter os dados originais. Através da análise crítica dos resultados obtidos para um *dataset* de imagens monocromáticas iremos propor um mecanismo eficiente para a compressão não-destrutiva de cada uma dessas Fontes.**

***Palavras-chave*—Compressão não-destrutiva de dados, Taxa de compressão, Imagem monocromática**

I. Introdução

Num mundo onde a tecnologia e o digital estão cada vez mais acessíveis e presentes na vida das pessoas, e em que praticamente todo o conteúdo multimédia criado está disponível na Internet, é inevitável reconhecer o papel fulcral que as técnicas de compressão de dados tiveram nesta revolução informática. Num computador toda a informação é representada em bits (0's e 1's), logo, todo o conteúdo digital que criamos ou consumimos consiste num conjunto de bits que está armazenado em memória. Se não forem comprimidos, o espaço ocupado por estes dados poderá ser enorme, tornando inviável o seu armazenamento ou mesmo a sua transmissão em rede.

Uma imagem *bitmap* monocromática, de dimensões 1280x720, que se traduz em 921600 pixéis, com um nível de quantização de 8 bits por pixel, ocuparia 900KB em disco. Para transmitir esta imagem através de um modem com 30Kbps de largura de banda, seriam necessários 4 minutos, uma longa espera. A compressão não-destrutiva desta imagem permitiria reduzir o número de bits utilizados para representar a informação, sem comprometer a sua integridade após descompressão e possibilitando assim uma transmissão mais rápida e um armazenamento mais eficiente. A eficiência de um algoritmo de compressão não-destrutivo pode ser avaliada de diferentes formas. Pode ser medida a complexidade de implementação do algoritmo, a memória necessário para o implementar, a rapidez de compressão e descompressão dos dados, ou a taxa de compressão obtida através desse algoritmo. Neste trabalho, o critério que teremos em consideração para a determinação daquela que achamos ser a melhor técnica de compressão não-destrutiva para cada fonte de dados de um conjunto de imagens monocromáticas, para contexto de armazenamento, será a maior taxa de compressão conseguida, dada pela seguinte fórmula:

$$TaxaCompr = \frac{Loriginal - Lcodificado}{Loriginal} \times 100\%$$

Em que $L_{original}$ é o número de bits necessário para representar cada amostra da fonte original e $L_{codificado}$ é o número médio de bits utilizado para codificar cada símbolo da fonte comprimida.

A compressão não-destrutiva de dados pode ser dividida em duas fases. A primeira fase consiste no pré-processamento e modelação da fonte de dados, enquanto a segunda fase compreende a codificação desses dados.

II. Fase de Pré-Processamento

O pré-processamento da fonte é um passo fundamental para a obtenção de uma maior taxa de compressão. Nesta fase, explora-se e transforma-se a informação da fonte com vista a aumentar a sua redundância e diminuir, consequentemente, a sua entropia, o que, para alguns dos algoritmos que se centram na utilização de códigos entrópicos e que passaremos a descrever na fase de codificação, possibilitará um aumento significativo da compressão dos dados transformados.

A. *Agrupamento de Símbolos*

Tal como o nome sugere, este algoritmo consiste num agrupamento *n* a *n* de símbolos. Geralmente é vantajoso estudar várias variáveis em simultâneo (agrupamentos), dado que permitem obter um valor de entropia inferior ao da fonte original.

B. *Run-Length Encoding*

O Run-Length Encoding (RLE) é um algoritmo simples que permite abreviar sequências de símbolos repetidos ao substituí-las por dois valores, o número de vezes que o símbolo se repete nessa sequência e o valor do símbolo. Esta abordagem é particularmente útil em fontes de dados cujos símbolos se repetem consecutivamente. A seguir é apresentado um exemplo de uma fonte (em cima) codificada com RLE (em baixo):

80	80	80	56	56	56	56	56	78

{80,3}	{56,5}	{78,1}

Figura 1: RLE [2]

Ao aplicar o RLE a fontes cujas amostras não se repitam com frequência obter-se-á um efeito contrário ao desejado.

C. *Move-to-Front*

O algoritmo Move-to-Front (MTF) começa por reconhecer e construir uma sequência com alfabeto de símbolos da fonte fornecida. Para uma imagem monocromática quantizada a 8 bits/símbolo esta sequência conterá os valores pertencentes a conjunto [0-255]. A fonte é então percorrida iterativamente. O output resultante de cada iteração representa o índice em que o símbolo lido se encontra na sequência. Esta sequência é então atualizada, sendo movido o símbolo lido, da posição sua posição atual para o topo da sequência. Este algoritmo reside na ideia de que o símbolo recente é o mais provável de ocorrer no futuro. Quando os dados possuem correlação local ao nível da ocorrência de símbolos, é possível obter um output cuja entropia é inferior à da fonte original. Costuma ser vantajoso utilizar este algoritmo após ter sido aplicada a transformada de Burrows-Wheeler (BWT) à fonte como veremos a seguir.

D. *Transformada de Burrows-Wheeler*

A Transformada de Burrows-Wheeler (BWT) traduz-se na aplicação de uma transformação reversível na fonte. Esta transformação é alcançada através da construção inicial de uma matriz que contém todas as permutações cíclicas da fonte recebida. Esta matriz é então ordenada lexicograficamente. O output a considerar consiste na última coluna da matriz obtida depois do ordenamento, juntamente com o índice da linha onde se encontra a fonte original nesta mesma matriz. Verifica-se que o tamanho do output resultante é superior ao da fonte original, no entanto, aplicando ao output deste algoritmo uma transformação como a MTF ou a RLE, é possível obter uma taxa de compressão consideravelmente superior, dado que o output da BWT exibe correlação local ao nível da frequência de ocorrência dos símbolos. É, portanto, vantajoso combinar esta transformação com outros algoritmos na fase de pré-processamento.

E. *Preditores*

Para obter uma codificação mais eficiente, a codificação preditiva recorre apenas ao contexto e ao histórico dos dados, maximizando probabilidades e, consequentemente, reduzindo a entropia. O valor de um pixel pode ser calculado a partir do valor de pixéis vizinhos. Um preditor combina um número de pixéis vizinhos de maneira a prever o próximo pixel, removendo, assim, parte da redundância espacial. O valor codificado é um resíduo (erros de previsão) calculado da seguinte forma:

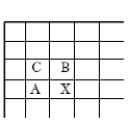
$$\text{resíduo} = \text{valor\_original} - \text{valor\_previsto}$$

Para que seja possível de descodificar a sequência de resíduos resultante da codificação, sem perdas de informação, é necessário conhecer o método utilizado para gerar a previsão.

a. *Preditores Lineares*

Os preditores lineares baseiam-se na ideia de que a dependência amostral é quantificável, ou seja, numa imagem em que o valor de um pixel dependa do valor do pixel anterior, será possível prever todos os restantes pixéis da imagem.

O codec JPEG Lossless recorre a um preditor linear que compara três pixéis vizinhos, A, B e C para prever o píxel X. A previsão do pixel X é feita de acordo com os seguintes preditores:



selection-value	prediction
0	no prediction
1	A
2	B
3	C
4	A+B-C
5	A+((B-C)/2)
6	B+((A-C)/2)
7	(A+B)/2

Figura 2-Filtro JPEG Lossless[7]

No codec de imagem PNG é realizada uma compressão entrópica de resíduos com base em modelos de previsão. À semelhança do JPEG Lossless, no PNG é explorada a correlação entre amostras consecutivas, ou seja, cada byte é previsto com base nos valores de bytes anteriores. Para efetuar a previsão é seguido o seguinte modelo de preditores (filtro):

Type	Name	Filter Function
0	None	`Filt(x) = Orig(x)`
1	Sub	`Filt(x) = Orig(x) - Orig(a)`
2	Up	`Filt(x) = Orig(x) - Orig(b)`
3	Average	`Filt(x) = Orig(x) - floor((Orig(a) + Orig(b)) / 2)`
4	Paeth	`Filt(x) = Orig(x) - PaethPredictor(Orig(a), Orig(b), Orig(c))`

```
p = a + b - c
pa = abs(p - a)
pb = abs(p - b)
pc = abs(p - c)
if pa <= pb and pa <= pc then Pr = a
else if pb <= pc then Pr = b
else Pr = c
return Pr
```

Figura 3- Filtro PNG[7]

b. *Preditores Não-Lineares*

Dada a elevada complexidade dos preditores não-lineares deixamos apenas a breve observação de que estes se baseiam essencialmente em redes neuronais e que geralmente permitem obter taxas de compressão bastante altas.

III. Fase de Codificação

Existem vários métodos de codificação. Entre eles destacamos a codificação de Huffman e Aritmética e a codificação por dicionário LZ77, LZ78 e LZW.

A. *Codificação de Huffman e Codificação Aritmética*

A codificação de Huffman e a codificação aritmética são codificações entrópicas, ou seja, cada símbolo é codificado com base na sua probabilidade estatística de ocorrência. Os códigos dos símbolos cujas frequências de ocorrência sejam superiores são representados com um menor número de bits comparativamente com os que ocorrem, na fonte, com uma menor frequência. Para estes dois algoritmos a fonte de dados a ser codificada é percorrida duas vezes; a primeira vez para contabilizar e registar o número de ocorrências de cada símbolo e a segunda para efetuar a codificação desses símbolos. A taxa de compressão obtida através destes algoritmos está limitada pela entropia associada à fonte.

B. *Codificação por Dicionário Adaptativo*

Nestes algoritmos é desnecessário conhecer, à priori, o modelo estatístico da fonte de dados. É usado um dicionário construído adaptativamente à medida que os dados são iterados. De notar que os dados serão depois descodificados

não-destrutivamente sem que haja o conhecimento do dicionário resultante da codificação.

a. LZ77

Este algoritmo explora a repetição de conjuntos de um ou mais símbolos na fonte. Através do conceito de janela deslizante, dividida em dois buffers, o *search* e o *look-ahead buffer*, a fonte é percorrida. A ideia é procurar no *search buffer* o maior padrão que ocorre no *look-ahead buffer* (início no primeiro símbolo). Cada ocorrência é então codificada da seguinte forma: *{offset; length; próximo símbolo},* em que o *offset* é a distância do *match pointer* ao *look-ahead buffer, o length* é o número de símbolos consecutivos no *search buffer* que coincidem com símbolos consecutivos no *look-ahead buffer*. No caso em que o símbolo lido *s* não se encontra no *search buffer*, este é codificado como {0,0,*s*}. A imagem seguinte, Figura 4, representa um exemplo de uma iteração do algoritmo LZ77. Desta iteração em particular resultará o tripleto {7,4,'r'} no lugar do conjunto de símbolos 'abrar' no *look-ahead buffer*.

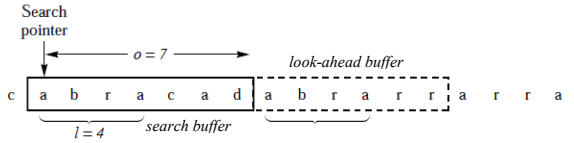


Figura 4-Processo de Codificação LZ77[4]

b. LZ78

Neste algoritmo os conjuntos de um ou mais símbolos são codificados com a notação {*i,c*}, em que i representa o índice no dicionário (*i*=0 indica que não foi encontrada nenhuma entrada para o símbolo *c* no dicionário) e *c* é um símbolo. Começa-se por criar um dicionário vazio. A fonte é percorrida e o dicionário é preenchido de forma explicita. Numa situação prática é necessário limitar o crescimento do dicionário para que este não cresça 'infinitamente', após ter atingido o limite de crescimento estabelecido, o dicionário torna-se fixo.

1. LZW

O LZW é uma variante do LZ78, utilizada no codec de imagem GIF. A ideia por detrás deste algoritmo reside em evitar o envio de duplos {*i,c*} ao enviar apenas o índice das entradas no dicionário. Para que isto seja possível é necessário começar por criar um dicionário com entradas para todos os símbolos do alfabeto da fonte a codificar. A fonte é percorrida e, à semelhança do LZ78, em cada iteração, é adicionada uma nova entrada ao dicionário. Tal como no LZ78 mantem-se a necessidade de limitar o crescimento do dicionário para evitar *overflows* ao codificar grandes fontes de dados. Exemplificando para uma sequência '010110', o resultado da codificação seria: 1 2 3 4 tendo sido gerado o seguinte dicionário durante a codificação:

Figura 5 - Dicionário LZW resultante

Índice	Entrada
1	0
2	1
3	01
4	10
5	011

IV. Técnicas de Compressão

Para obtermos taxas de compressão superiores devemos explorar e aplicar, de forma combinada, os algoritmos de transformação e codificação que melhor se adequam à estrutura da fonte que se pretende comprimir. Existem várias técnicas de compressão amplamente utilizadas atualmente, que recorrem à aplicação de alguns dos algoritmos descritos anteriormente neste artigo, por uma ordem que lhes confere taxas de compressão especialmente eficientes. A seguir mencionamos algumas dessas técnicas, não deixando de referir que podem e devem ser exploradas, para cada fonte de dados em particular, outras técnicas mais personalizadas que se possam adequar melhor à fonte de dados a comprimir.

A. Deflate

A técnica de compressão Deflate combina a codificação LZ77 com a codificação de Huffman. Nesta técnica a fonte de dados é dividida em blocos. O LZ77 é aplicado a cada bloco. São então calculados os códigos de Huffman para os símbolos de cada bloco, dada a ocorrência de cada símbolo dentro de cada bloco. Esta técnica tende a ser pouco eficiente na compressão de fontes que exibam poucas ou nenhumas repetições. Esta é a técnica é utilizada no compressor *gzip, no ZIP* e no codec de imagens PNG (na compressão de resíduos).

B. bzip2

A técnica de compressão *bzip2* começa por transformar os dados aplicando a transformação de RLE à fonte original. É aplicada então a BWT seguida de MTF. Ao resultado da MTF é aplicada novamente RLE. A partir daqui é aplicada a codificação de Huffman com algumas variações.

C. LZMA

LZMA é uma versão melhorada do LZ77, que utiliza cadeias de Markov. As cadeias de Markov modelam sequências através de probabilidades de transição de estados (símbolos). Nesta técnica são, inicialmente, usados preditores lineares, mais concretamente, o filtro Delta. O output deste filtro é codificado através do conceito da janela deslizante do LZ77. O resultado desta codificação é então codificado através de um algoritmo de codificação entrópica conhecido como *Range Encoding* que se baseia na codificação aritmética. O LZMA é uma das técnicas de compressão e descompressão incluídas no formato 7zip.

V. Especificidades

Para determinarmos o conjunto de técnicas de compressão a explorar para cada imagem começámos por obter e analisar a distribuição estatística dos símbolos para cada fonte e a respetiva Entropia, que consiste no limite mínimo teórico para o número médio de bits por símbolo que é possível obter através de um codificador entrópico.

Ficheiro	Tamanho [MB]	Entropia [bits/símbolo]	Tamanho Mínimo c/ Codificador Entrópico [MB]
pattern.bmp	45.7810	1.8292	10.4678
egg.bmp	16.9166	5.7242	12.1043
zebra.bmp	15.9618	5.8312	11.6346
landscape.bmp	10.4955	7.4205	9.7352

Tabela 1 - *Dataset* original (dados não tratados)

A partir da Entropia e do tamanho original dos ficheiros, calculámos o tamanho mínimo teórico possível de obter através da aplicação de um codificador entrópico aos dados sem qualquer transformação ou codificação adicional, para cada imagem. Cada pixel nas imagens do *dataset* original, ocupa, originalmente, sem compressão, 8 bits em memória, daí temos que o limite máximo para a Entropia é de 8 bits/símbolo. Tal dar-se-ia se a ocorrência de cada símbolo, na Fonte, fosse equi-provável, ou seja, quanto mais uniforme for o modelo estatístico da Fonte, mais alta será a sua Entropia. Através da interpretação destes dados e partindo da observação empírica de cada imagem inferimos que seria interessante explorar a aplicação de transformações à Fonte de dados de 'pattern.bmp' que tirassem partido da repetição de símbolos e do facto desta imagem possuir, em 256 símbolos do seu alfabeto, 2 símbolos, o pixel de cor branca e o pixel de cor preta, que ocorrem de forma distintamente predominante. Destas possíveis transformações destacamos o Run-Length Encoding e o Agrupamento de símbolos. Em termos de distribuição estatística a 'egg.bmp' e a 'zebra.bmp' apresentam alguma semelhança, no entanto visualizamos que a disposição dos símbolos varia significativamente de uma para a outra. Daqui acreditamos que possa demostrar-se vantajosa a transformação da Fonte de 'zebra.bmp' recorrendo a preditores lineares, mais concretamente ao filtro Up do codec PNG do qual resultarão resíduos cuja redundância poderá ser explorada com o Run-Length Encoding ou com uma combinação da Transformada de Burrows-Wheleer seguida de Move-to-Front. Já para a 'egg.bmp' seria interessante aplicar o RLE dada a repetição de pixéis pretos na imagem. Observando o histograma do ficheiro 'landscape.bmp', constatamos que existe maior dispersão de ocorrência de símbolos, comparativamente aos outros ficheiros do *dataset*, o que se traduz num valor de Entropia mais elevado que, em termos práticos, é igual ao limite máximo (8 bits/símbolo) para este tipo de imagens. O agrupamento de símbolos permitirá diminuir a Entropia desta Fonte.

Depois da fase de Transformação das Fontes, será testada a aplicação dos diversos algoritmos de codificação por dicionário adaptativo e codificação entrópica mencionados neste artigo. Habitualmente, dependendo das especificidades da Fonte, pode ser vantajoso utilizar, por exemplo, uma técnica de dicionário adaptativo seguida pela codificação entrópica desse output. Embora possamos seguir um raciocínio lógico, com base no conhecimento que possuímos dos diversos algoritmos que apresentamos neste artigo, para prevermos qual seria a melhor técnica de transformação e codificação a aplicar, dada as especificidades de cada imagem, convém frisar que é de nosso maior interesse avaliar o desempenho desta e de outras técnicas de compressão, relacionando a metodologia da técnica com as especificidades da Fonte à qual foi aplicada.

VI. Resultados experiementais e análise

Para obtermos todos os resultados experimentais apresentados e analisados neste artigo, implementámos um programa, na linguagem de programação *python,* que reúne um conjunto de algoritmos e técnicas de compressão e que nos permitiu, de forma compacta, comprimir todos os ficheiros do *dataset.* Como referência de comparação para o nosso trabalho, utilizámos os valores de compressão, das imagens disponibilizadas, resultantes do codec PNG.

Lossless Codec	**Compressed File Size Reference Value [Mbytes]**			
PNG	pattern.bmp	egg.bmp	zebra.bmp	landscape.bmp
	2.1700	4.4100	5.2100	3.1700
Compression Rate:	95.26 %	73.93 %	67.36 %	69.80 %

Tabela 2 - Valores de Referência Codec PNG

Como ponto de partida, começámos por comprimir todos os ficheiros através da aplicação, individual, dos algoritmos de codificação, sem efetuar qualquer tipo de transformação na Fonte.

Compression Algorithms	**Resulting Compressed File Size [Mbytes]**			
	pattern.bmp	egg.bmp	zebra.bmp	landscape.bmp
Huffman	11.0292	12.1446	11.7006	9.7762
LZ77	9.9155	12.2005	12.7381	9.4299
LZ78	2.7301	7.5852	8.0446	6.1309
LZW	2.7068	7.6645	8.7883	5.5514

Tabela 3-Resultados da codificação sem transformação da Fonte

Analisando a Tabela 1 e a Tabela 3 constatamos que o algoritmo de codificação por dicionário adaptativo, LZ78 e LZW, permite obter uma taxa de compressão significativamente superior ao de qualquer codificador entrópico. Na Tabela 4, seguinte, apresentamos alguns dos resultados experimentais mais relevantes que obtivemos a partir da combinação dos vários algoritmos da fase de pré-processamento com os da fase de codificação.

Compression Techniques	**Resulting Compressed File Size [Mbytes]**			
	pattern.bmp	egg.bmp	zebra.bmp	landscape.bmp
Sub+Huffman	13.3982	8.5699	9.2285	5.6279
Up+Huffman	13.5036	8.7164	9.1541	5.5220
Grouping+Huffman	11.0278	12.1404	11.6996	9.775
Sub+Grouping+Huffman	13.9375	10.4527	11.3261	7.3349
Up+Grouping+Huffman	13.9570	10.7104	11.1136	7.2412
RLE+Grouping+Huffman	3.8933	14.1100	14.2661	8.5214
RLE+Huffman	3.8933	14.1100	14.2661	8.5214
Sub+LZ78	3.5126	7.3138	8.2691	5.1282
Up+LZ78	3.5793	7.5315	8.2180	5.0572
LZW+Huffman	2.6861	7.6023	8.6624	5.5447
LZ78+Huffman	2.5197	7.3558	7.8157	6.0944
Up+LZW+Huffman	3.5235	7.0893	8.0476	4.8103
Up+LZ78+Huffman	3.2683	6.9146	7.5868	4.5484
Sub+LZ78+Huffman	3.2020	6.7075	7.6436	4.6184
LZMA	1.8416	4.5965	5.4003	3.0757
Up+LZMA	2.4458	4.6845	5.5805	2.6085
Sub+LZMA	2.4594	4.5299	5.574	2.6971
Up+Grouping+LZMA	2.6228	4.7445	5.6989	2.6398
Sub+Grouping+LZMA	2.6671	4.5712	5.7435	2.7405
Grouping+LZMA	1.8982	5.1897	5.6360	3.1183
RLE+LZMA	1.6704	4.9690	5.9131	3.6452
RLE+Grouping+LZMA	1.6679	4.9698	5.9130	3.6441
Up+RLE+LZMA	2.0339	5.0645	5.8679	3.1195
Sub+RLE+LZMA	2.0652	4.8495	5.9370	3.1285
BWT+MTF+LZMA	3.5492	8.9566	12.2724	5.5204
BWT+RLE+LZMA	3.2231	8.9295	11.7958	5.6203
gzip	2.2729	6.2484	7.2093	4.2420
Sub+gzip	3.3598	5.9952	7.4549	4.0679
Up+gzip	3.3233	6.2485	7.3237	4.0613
RLE+gzip	2.4228	7.6912	9.1109	5.3993
Grouping+gzip	2.3128	6.9650	7.5492	4.4473
BWT+RLE+gzip	5.0037	11.8649	15.9117	7.5079
bzip2	1.7243	4.5271	5.3641	3.3327
Sub+bzip2	2.0319	4.6097	5.6463	3.0687
Up+bzip2	2.0113	4.8247	5.6070	3.0773
Grouping+bzip2	1.7830	5.0316	5.7443	3.3785
Sub+Grouping+bzip2	2.0189	4.4868	5.5158	2.9902
Up+Grouping+bzip2	2.0007	4.6957	5.4759	2.9904

Tabela 4-Resultados Experimentais

Antes de procedermos com a análise dos resultados experimentais deverão ser tidos em consideração alguns aspetos:

a. A técnica Deflate foi conseguida através do *gzip;*

b. O algoritmo de Run-Length Encoding foi desenvolvido com a capacidade de executar a transformação tanto ao nível das linhas (horizontais) como das colunas (verticais) e de escolher qual destas opções permite obter uma compressão mais eficiente. A distinção desta escolha é feita através da leitura do formato do ficheiro resultante do algoritmo que varia entre '.rle' e '.rleT', o que possibilita a posterior descompressão do ficheiro.

c. Ao colocar-mos em prática a Transformada de Burrows-Wheeler apercebemo-nos de que esta seria impraticável em Fontes cuja quantidade de símbolos fosse tão ou mais elevada que as do *dataset* que estamos a processar. Daqui surgiu a necessidade de segmentar a fonte, em blocos de tamanho considerável, e de aplicar a cada um, separadamente, a transformada, contornando assim o problema de memória levantado.

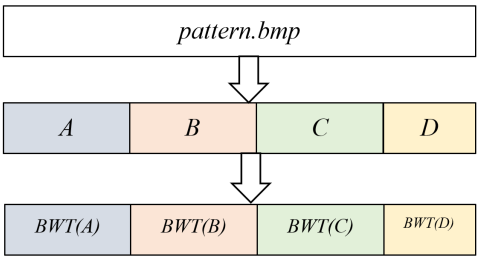
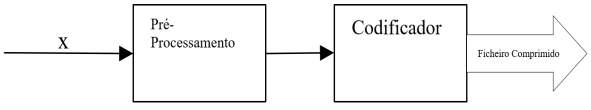


Figura 6-Transformada de Burrows-Wheeler em Blocos

d. A transformação da Fonte na fase de pré-processamento, dependendo do algoritmo e da forma como é aplicado pode aumentar de forma considerável o tamanho em bytes ocupado pela Fonte transformada, em memória. Por exemplo, os resíduos resultantes da aplicação do preditor Sub ou Up a uma imagem quantizada a 8 bits, são representados com 16 bits, logo não haveria qualquer interesse em armazenar os resíduos como resultado da compressão, no entanto, como já foi referido, o objetivo destas transformações passa, principalmente, por aumentar a redundância da Fonte e não, propriamente, diminuir o seu espaço ocupado em memória, logo, a fase de pré-processamento é sempre seguida da fase de codificação de acordo com o seguinte esquema:



e. O algoritmo 'Grouping' consiste no agrupamento de símbolos 2 a 2 executado na fase de pré-processamento.

Ao analisar-mos os resultados obtidos percebemos que a inferência posteriormente feita em relação à aplicação de preditores lineares, mais concretamente, Up ou Sub, na fase de pré-processamento da imagem 'zebra.bmp' não é tão eficiente como acreditávamos que pudesse ser. Este facto deve-se à existência de transições de cor mais abruptas, que podemos visualizar na imagem. No entanto, comparando os resultados da aplicação da codificação de Huffman com os de Up+Huffman ou Sub+Huffman, verificamos que a utilização do modelo Up ou Sub na imagem 'zebra.bmp' permitiu reduzir a Entropia veiculada na Fonte. Por outro lado, se tivéssemos um preditor linear baseado num modelo matemático que modelasse as curvas características do pelo de uma zebra, teríamos conseguido um resultado para o valor de Entropia muito inferior ao conseguido com os modelos mais genéricos Up e Sub. O tipo das transições de cor presentes numa imagem é o fator que determina o grau de eficiência da aplicação destes preditores lineares. Em imagens fotográficas naturais, as transições de cor tendem a ser suaves, ou seja, existe contexto entre pixéis e é exatamente esta correlação existente entre pixéis vizinhos que os preditores lineares exploram. Feita esta observação concluímos que os resultados obtidos aquando da aplicação de preditores lineares nas imagens do *dataset* estão de acordo com o que seria de esperar; a Entropia veiculada na Fonte para as imagens 'egg.bmp', 'landscape.bmp' e 'zebra.bmp' diminui aquando da execução dos modelos Up ou Sub já que estas imagens consistem em imagens foto-realísticas cujas transições de cor são suaves, enquanto que para a imagem 'pattern.bmp' a Entropia aumenta por ser complicado tirar partido da dependência estatística da Fonte por esta apresentar transições abruptas de cor, daí não haver contexto.

O agrupamento de símbolos 2 a 2 diminuiu a Entropia de todas as fontes como seria de esperar, contudo verifica-se que a utilização de preditores lineares é mais eficiente a reduzir a entropia, para as três imagens foto-realísticas do *dataset*, do que o Grouping. Para o 'pattern.bmp' confirma-se o oposto.

Ao comparar-mos o tamanho resultante da aplicação individual da codificação de Huffman ao ficheiro 'pattern.bmp' com o tamanho obtido da aplicação do RLE+Huffman, percebemos que o Run-Length Encoding tem a capacidade de reduzir imenso o tamanho do ficheiro que é caracterizado, originalmente, por apresentar várias repetições de símbolos, característica esta que o RLE explora de maneira eficaz. Esperávamos obter um melhor desempenho do RLE para a imagem 'egg.bmp' tendo em conta a repetição de pixéis de cor preta. Esta falha poderá ser justificada pelo facto de existir uma grande dispersão de símbolos utilizados para representar os dois ovos centralizados na imagem. Assim o RLE torna-se uma desvantagem envés de uma mais valia dado que todos os símbolos que não se repetem de forma consecutiva passarão a ser codificados com 2 símbolos envés de 1. O RLE tem um mau desempenho em imagens fotográficas naturais dado que estas apresentam transições suaves de cor, logo é difícil encontrar repetições suficientes de símbolos que compensem pela ocorrência de símbolos individuais que passam a ser representados por um par de símbolos, tal como concluímos em 'egg.bmp'.

A transformada de Burrows-Wheeler foi combinada com a Move-to-Front ou com RLE e testada para os vários ficheiro, no entanto demonstrou ficar aquém daquilo que seria de esperar, talvez pela implementação em blocos utilizada. Como iremos analisar mais à frente, a combinação RLE+BWT+MTF+RLE do bzip2 demonstrou ser bastante eficiente.

Como verificámos pela Tabela 3, o algoritmo LZ78 ou a sua variante LZW permitiram obter uma taxa de compressão

bastante superior à de qualquer codificador entrópico quando aplicado ao *dataset*. Daqui e pelos resultados apresentados na Tabela 4 verificámos que com a combinação de LZW+Huffman ou LZ78+Huffman conseguimos obter uma taxa de compressão maior para todos os ficheiro, comparativamente com as taxas de compressão obtidas das combinações já analisadas que incluíam o RLE,Sub ou Up e/ou Grouping na fase de pré-processamento e Huffman na fase de codificação. Podemos olhar para o LZW como um agrupamento de símbolos que cresce exponencialmente sem alterar o Alfabeto daí ser muito mais eficiente que o Grouping para este *dataset.* Tanto o LZ78 como o LZW apresentam resultados tão bons para o 'pattern.bmp' por existirem bastantes repetições de símbolos e por ocorrerem, na imagem predominantemente apenas 2 símbolos. Para os restantes 3 ficheiros conseguimos obter uma taxa de compressão ainda melhor utilizando preditores lineares (Up ou Sub, dependendo da imagem) antes de executar o LZ78+Huffman. O LZ77 executado isoladamente não teve a melhor prestação quando comparado com os outros algoritmos de codificação por dicionário adaptativo, mas, aquando da sua utilização na técnica Deflate (*gzip*), em que o LZ77 é aplicado em blocos, foi possível obter uma melhor taxa de compressão que a da técnica LZW+Huffman ou LZ78+Huffman.

Explorando técnicas mais modernas como o LZMA ou o *gzip* conseguimos obter resultados ainda melhores. No caso do *gzip* é importante notar que a melhor taxa de compressão para 'pattern.bmp' foi obtida através da execução do *gzip* individualmente, sem qualquer transformação complementar da Fonte, pelo facto do modo de funcionamento do Deflate passar pela segmentação da Fonte. Para o 'egg.bmp' e para o 'landscape.bmp', o melhor resultado utilizando o Deflate foi conseguido utilizando o Sub e o Up no pré-processamento das Fontes tal como já tínhamos percebido ser vantajoso para estes ficheiros, pelos resultados analisado anteriormente. Para o 'zebra.bmp' não foi possível aumentar a taxa de compressão através da combinação de preditor linear com *gzip,* este facto deve-se também ao funcionamento, em blocos, do Deflate. Para a imagem 'pattern.bmp' a execução da técnica RLE+Grouping+LZMA foi a que resultou na melhor taxa de compressão registada para este ficheiro, ultrapassando mesmo a taxa de compressão conseguida com o Codec PNG. O facto desta imagem possuir 2 símbolos predominantes e de o LZMA se basear em cadeias de Markov representa um fator importante no sucesso desta técnica que tira partido da dependência estatística da fonte, e que modelam sequencias através da probabilidade de transição de estados. Foi através da técnica Up+LZMA que conseguimos obter a melhor taxa de compressão para 'landscape.bmp' neste trabalho. A imagem 'landscape.bmp' e a 'pattern.bmp' diferem imenso uma da outra a nível do modelo estatístico, no entanto percebemos que o Up, juntamente com o preditor linear Delta utilizado no LZMA são bastante eficientes a reduzir a entropia de 'landscape.bmp'. Para os outros dois ficheiros obtivemos também bons resultados dado que geralmente o LZMA é uma boa técnica de compressão. No entanto, os melhores resultados obtidos para a compressão de 'egg.bmp' e de 'zebra.bmp' foram conseguidos através do bzip2 embora não tenhamos conseguido ultrapassar, por pouco, o codec PNG na compressão destas imagens. A grande mais valia do bzip2 está na técnica, extremamente eficiente, que este utiliza na fase de pré-processamento e que consiste em RLE+BWT+MTF+RLE. Visto que o Sub e o Grouping resultaram bem no pré-processamento da imagem 'egg.bmp', a combinação destes algoritmos com os do bzip2 permitiram obter ainda uma melhor compressão do que utilizando apenas o Sub+Grouping+Hufman, como já havíamos testado. Daí darmos especial enfâse ao elevado valor da técnica de pré-processamento do bzip2, relembrando que na fase de codificação é aplicada a codificação de Huffman.

Abaixo apresentamos uma síntese dos melhores resultados conseguidos para a compressão de cada uma das imagens monocromáticas.

Compression Technique	**Resulting Compression Rate [%]**			
	pattern.bmp	egg.bmp	zebra.bmp	landscape.bmp
RLE+Grouping+LZMA	96.36	70.62	62.96	65.28
Sub+Grouping+bzip2	95.59	73.48	65.44	71.51
bzip2	96.23	73.24	66.39	68.25
Up+LZMA	94.66	72.31	65.04	75.15

Tabela 3-Taxas de Compressão Resultantes

Na Figura comparamos a taxa de compressão para cada ficheiro, obtida com as técnicas da Tabela 3, com as taxas de compressão conseguidas com o Codec PNG.

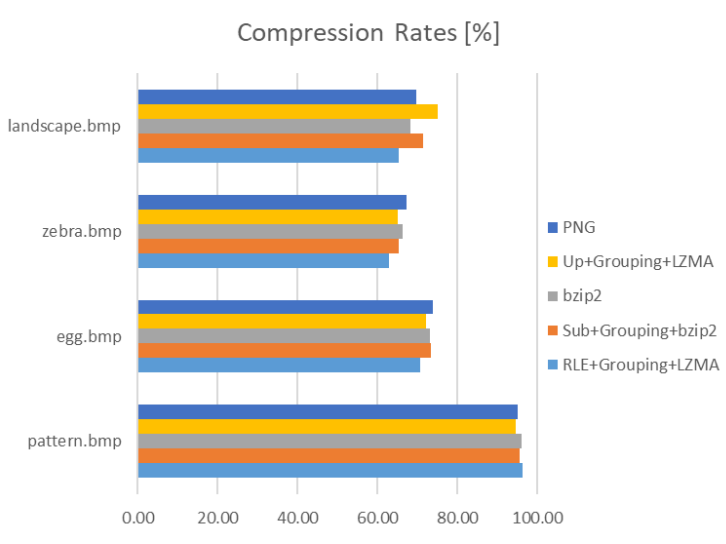


Figura 8-Comparação dos Resultados com Codec PNG

A partir destes resultados sugerimos a utilização da técnica RLE+Grouping+LZMA para a codificação de 'pattern.bmp', a Sub+Grouping+bzip2 para 'egg.bmp', o bzip2 para comprimir 'zebra.bmp' e a técnica Up+LZMA para a imagem 'landscape.bmp'.

Todos os dados e resultados experimentais analisados neste artigo encontram-se disponíveis, na integra, no ficheiro 'ResultadosExperimentais.xlsx' para leitura e/ou processamento.

VII. Conclusão

Neste trabalho foram apresentados vários algoritmos de compressão não-destrutiva para imagens monocromáticas. Foi analisado um *dataset* de 4 imagens em tons de cinza e exploradas várias soluções para a compressão eficiente de cada ficheiro. Implementámos e combinámos os diversos algoritmos utilizados na fase de pré-processamento e de codificação para serem aplicados aos ficheiros, com vista a comprimi-los. Registámos e analisámos os resultados, relacionando-os com as

especificidades de cada imagem. A partir desta interpretação, expusemos a técnica de compressão não-destrutiva que permitiu obter a taxa de compressão mais elevada para cada um dos ficheiros. Destacamos a existência de outras técnicas alternativas às sugeridas para cada imagem, que permitem também obter boas taxas de compressão.

REFERENCES

[1] Gupta, A. Bansal, V. Khanduja, Modern Lossless Compression Techniques: Review, Comparison and Analysis, Second International Conference on Electrical, Computer and Communication Technologies (ICECCT 2017)

[2] Ms. Ijmulwar, D. Kapgate, A Review on - Lossless Image Compression Techniques and Algorithms, International Journal of Computing and Technology, Volume 1, Issue 9, October 2014

[3] Lina J., K. (2009). Lossless Image Compression. In The Essential Guide to Image Processing (1st ed.). Elsevier. https://doi.org/10.1016/B978-0-12-374457-9.00016-0

[4] Khalid Sayood, Introduction to Data Compression (3rd ed.). Elsevier.

[5] Senthil Shannmugasundaram, Robert Lourdusamy, A Comparative Study of Text Compression Algorithms

[6] Prof. Dr. Rui Pedro Paiva, Powerpoint de apoio à cadeira de Multimédia, Imagem Digital

[7] Aleksej Avramović, Lossless Compression of Medical Images Based on Gradient Edge Detection